D1.4

Second release of MAX software: report on lighthouse codes' deployment at exascale

Luigi Genovese, F. Affinito, A. Alcaraz, O. Baseggio, L. Bellentani, C. Cardoso, I. Carnimeo, P. Delugas, F. Ferrari Ruffino, A. Ferretti, A. García, P. Giannozzi, R. Grima, J. Gutiérrez Moreno, M. Ippolito, G. Michalicek, M. Montagna, S. Orlandini, F. Paleari, F. Pedron, D. Sangalli, G. Sesti, N. Spallanzani, D. Varsano, F. Vaverka, N. Wittemeier, D. Wortmann, S. Baroni

Due date of deliverable	30/06/2025 (**month 30**)
Actual submission date	30/06/2025
Final Version	16/05/2025
Lead beneficiary	CEA (participant number 5)
Dissemination level	PU - Public

Document information

Project acronym	MAX
Project full title	Materials Design at the Exascale
Research Action Project type	Centres of Excellence for HPC applications
EuroHPC Grant agreement no.	101093374
Project starting/end date	01/01/2023 (month 1) / 31/12/2026 (month 48)
Website	http://www.max-centre.eu
Deliverable no.	D1.4

Authors	Luigi Genovese, F. Affinito, A. Alcaraz, O. Baseggio, L. Bellentani, C. Cardoso, I. Carnimeo, P. Delugas, F. Ferrari Ruffino, A. Ferretti, A. García, P. Giannozzi, R. Grima, J. Gutiérrez Moreno, M. Ippolito, G. Michalicek, M. Montagna, S. Orlandini, F. Paleari, F. Pedron, D. Sangalli, G. Sesti, N. Spallanzani, D. Varsano, F. Vaverka, N. Wittemeier, D. Wortmann, S. Baroni

To be cited as	Genovese et al. (2025): Second release of MAX software: report on lighthouse codes' deployment at exascale. Deliverable D1.4 of the HORIZON-EUROHPC-JU-2021-COE-01 project MAX (final version as of 29/06/2025). EC grant agreement no: 101093374, CEA, Grenoble, France.

Disclaimer

Contents

Executive Summary

This document presents the second release report of the MAX lighthouse codes in the current phase of the MAX Centre of Excellence. It follows from Deliverable D1.2, which laid the groundwork with the first set of production-ready, portable, and exascale-prepared versions of the core codes, and now documents the transition to innovative, disruptive scientific applications at the exascale frontier.

In this release, the focus shifts towards demonstrating how the MAX codes evolve and adapt to real-world scientific grand challenges when deployed on state-of-the-art HPC architectures. Across the different codes – BigDFT, FLEUR, QUANTUM ESPRESSO, SIESTA, and YAMBO – this document highlights:

- Substantial performance and scalability improvements, including GPU acceleration and advanced numerical interfaces;
- The adoption of modern programming models (OpenACC in QUANTUM ESPRESSO, improved solver abstraction in FLEUR, concurrent CUDA-Fortran, OpenACC, and OpenMP backends in YAMBO);
- Integration of new features to support research in fields such as phononics, excitonics, electrochemistry, and nanostructured materials (SIESTA, YAMBO),
- The development of non-trivial scientific workflows exploiting hybrid quantum/-classical dynamics at scale (BigDFT),
- Systematic efforts for deployment and validation across major EuroHPC platforms, ensuring reproducibility, portability, and user support.

A consistent emphasis is placed on the algorithmic innovations and software engineering practices needed to achieve exascale performance and robustness. In parallel, each team reflects on open challenges and outlines strategies for future development.

This report is structured on a code basis, allowing each team to independently present their achievements and challenges, despite the work is the result of integrated discussions and input from HPC centres at the CoE level. A concluding section provides a synthesis of the overall progress and a forward-looking perspective on the roadmap to exascale-era scientific discovery.

1 Introduction

This document presents the second release report of the MAX Centre of Excellence lighthouse codes, reflecting the ongoing progress in preparing key electronic-structure applications for the exascale era. Building on the foundations laid in earlier stages of the project, this deliverable continues to focus on performance improvements (including portability), software engineering, and scientific capabilities, as the codes adapt to the evolving landscape of high-performance computing.

Each code team has worked to enhance computational efficiency, improve portability across different architectures, and extend functionality to address new scientific challenges. As the codes are deployed and tested on a wide range of EuroHPC platforms, this release marks a step forward in aligning software development with the demands and opportunities of exascale systems.

The structure of the report follows a code-by-code format, allowing each team to highlight technical achievements, challenges encountered, and areas of scientific application. A concluding section provides a summary of collective progress and outlines the direction for future work within the MAX framework.

2 QUANTUM ESPRESSO

2.1 Release qe-7.5

We present here the contents of `qe-7.5` QUANTUM ESPRESSO version, released at the end of June 2025 –month 30 of the MAX third phase. This release consolidates the coding work from previous `qe-7.4.x` versions, which has focused on removing `CUDA-Fortran` constructs in favour of `OpenACC` and reorganising the local parallelism for the GPU offloaded version. This has brought three main advantages for users and developers:

- Significant speed-up for medium and large jobs executed on CUDA GPUs
- More unified Fortran sources with none or very few logical branching between accelerated and host-executed builds.
- Fully unified logic between the CUDA offloaded version and the one offloaded with OpenMP target

About the last point, it is important to stress that the completion of the logical unification of the `KS_Solvers` is instrumental to the introduction of the refactored band parallelization.

New features

This new release also brings about some new feature, mostly provided by the community but with essential effort by the core developer group for the integration of local parallelism and for the interoperability. The most worth of notice among these new features are reported below.

New Molecular dynamic functionalities. This version features the first stable results of the ongoing work for unifying the molecular dynamics functionalities of the suite into a unified framework. The current version of `pw.x` can use all the thermostats and barostats that were already available for `cp.x`. One notable addition introduced by the community is a new engine for performing path integral molecular dynamics. This significant new feature is also important because it demonstrates the presence of a very active community in this field, not only of users but also developers.

Advanced DFT+U methodologies. Two new DFT+U variants have been introduced through a significant contribution from the MaX group at Bremen University: the constrained DFT+U method and the orbital-resolved DFT+U method. These additions expand the existing suite of related computational approaches. The core developer group provided essential support for enhancing offloading capabilities across all DFT+U methods. Additionally, these methodologies generate sophisticated XML output that can be optimised using machine learning techniques, as demonstrated in WP2 (deliverables D2.4 and D2.5). This XML input/output functionality serves as a crucial interface between workflow managers and machine learning analysis tools.

Two chemical potential method. This method provides a constrained-DFT scheme for computing structural and vibrational properties of photo-excited insulators.

Reserved and beta features

Some features are currently reserved for testers and scientific collaborators. Although development is complete, these features have been placed in separate branches of `qe-7.5` for either beta testing or due to temporary scientific embargoes. These include:

- **Improved magnetic SCF.** This feature allows the mixing of total charge density and total magnetisation to be decoupled, improving the stability and efficiency of the SCF loop for challenging magnetic systems, while remaining effective for systems that are easy to converge. It is currently available on a branch of this `GitLab` repo. [1]
- **Excited states' energy gradients.** This feature extends the `turbo_davidson` application to compute energy gradients of excited states. A detailed example is provided in a dedicated subsection below. Due to a temporary scientific embargo, the code is hosted in a private `GitLab` repository.[2]

Build Systems, Package Managers, and Deployment

Supported Architectures. QE-7.4.1 and the development branch QE-7.5 have been deployed and validated on major EuroHPC clusters, supporting the following architectures:

- Intel and AMD `x86_64` CPUs,

[1] https://gitlab.com/pietrodelugas/q-e/-/tree/new_magnetic_mix
[2] https://gitlab.com/Baseggio/q-e/-/tree/qe_Z

- NVIDIA Grace Superchips,
- Generic ARM processors,
- Fugaku ARM processors,
- CUDA GPUs, with testing conducted on A100, H100, and H200 models.

For AMD GPUs, a variant of QE-7.4.1 has been released and tested using the `CCE-15` toolchain. This release is available in the official QE repository. [3] Porting and adaptations for QE-7.5 are currently underway using the `CCE-19` toolchain, and are available in a dedicated `GitLab` repository. [4]

Build Systems. Both `autoconf` and `CMake` build systems remain fully supported and functional across all toolchains. Ongoing work continues to incrementally enhance compatibility with additional toolchains. Recent improvements include full support for the `AOCC` toolchain, including automatic recognition of the `blis` and `flame` libraries. Starting from version QE-7.4.1, the `autoconf` system includes a dedicated `arch` option tailored for the `Cray` computing environment, streamlining configuration and setup.

Package Managers. QE is distributed via comprehensive sets of recipes for both `Spack` and `EasyBuild`. The current release has been prepared to ensure that existing package recipes can be updated seamlessly to reflect the new version. After this general overview of the release, we dedicate the remainder of this QE to brief subsections that highlight specific aspects of the release.

2.2 OpenACC refactoring and improved local parallelism for GPUs

We are nearing completion of the refactoring process to transition from legacy CUF (CUDA Fortran) code to OpenACC, significantly reducing code duplication. Figure 1 illustrates this transition: the upper panel shows the reduction in duplicated GPU code, while the bottom panel displays the shift in kernel types—highlighting the decline of CUF kernels and the corresponding increase in OpenACC kernels. Importantly, the refactored code, now fully based on OpenACC directives, offers improved compatibility with the `develop_omp5` branch. This advancement paves the way for a smoother and faster integration between the two primary development branches of the QUANTUM ESPRESSO suite: `develop` and `develop_omp5`.

The observed reduction in GPU code lines between release 7.4.1 and the `7.5-rc` branch (Figure 1) stems largely from extensive optimisation within the `KS_Solvers` library of QUANTUM ESPRESSO . Specifically, the RMM-DIIS and CG drivers for solving Kohn-Sham equations—both at the Γ-point and for general k-points—were fully redesigned. Associated routines for orbital manifold rotations were also rewritten.

This effort involved replacing redundant and legacy CUDA Fortran segments with modern OpenACC directives. The interfaces of these routines were streamlined to allow for simpler and more consistent invocation from both host and device contexts. This

[3] https://gitlab.com/QEF/q-e-omp-repository/-/releases/qe-7.4.1omp
[4] https://gitlab.com/fabrizio22/q-e/-/tree/develop_omp5_cce19

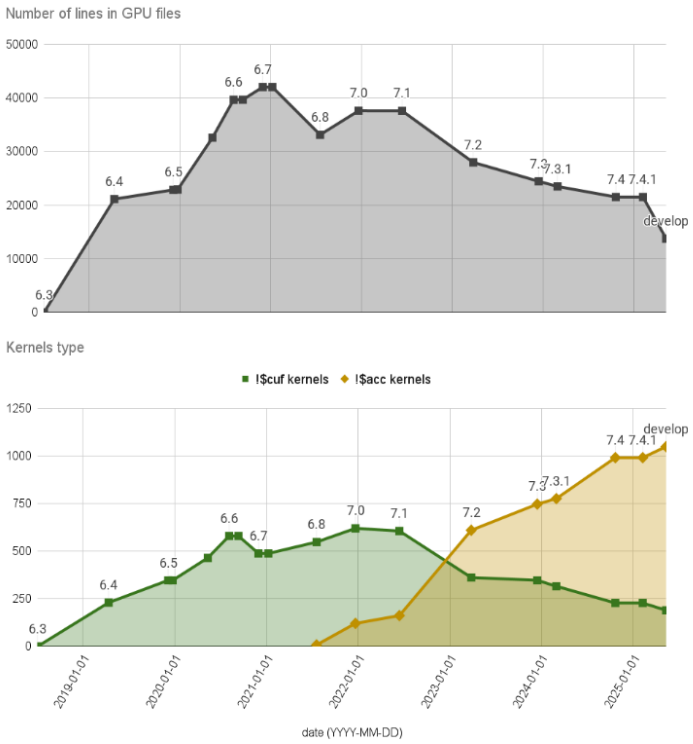

Figure 1: Number of lines in duplicated GPU files (upper panel) and kernel types (lower panel) in the QUANTUM ESPRESSO suite (updated on May 21st, 2025).

restructuring not only improves current maintainability but also establishes a solid foundation for future OpenMP extensions.

2.3 Parallel efficiency: Optimised communications libraries and extended adaptive FFT batching.

Communications within the FFTXlib library represent a significant portion of the total wall time in a QUANTUM ESPRESSO simulation. For this reason, FFTXlib was selected as a test case to evaluate the impact of different communication libraries on overall code performance. As described in Sect. 4.2.1 of D1.3, batching is essential to fully exploit available communication bandwidth, since individual FFT buffers are typically small in Plane-Wave DFT codes (grid sizes rarely exceed 600^3) relative to current hardware capabilities.

A basic hard-coded batching scheme was already implemented in QE. Our work focused on two key improvements. First, we extended this to an adaptive batching scheme, which more closely matches the peak collective bandwidth achievable in FFTXlib all-to-all communications (see Sect. 4.2.1 of D1.3). Second, we significantly expanded the use of FFT batching within QUANTUM ESPRESSO , introducing it into previously unoptimized code paths. This addressed communication bottlenecks—particularly in routines involving the meta-GGA potential, the local potential in the non-collinear case, and the band-sum.

As a representative application, we considered a water system of 471 molecules

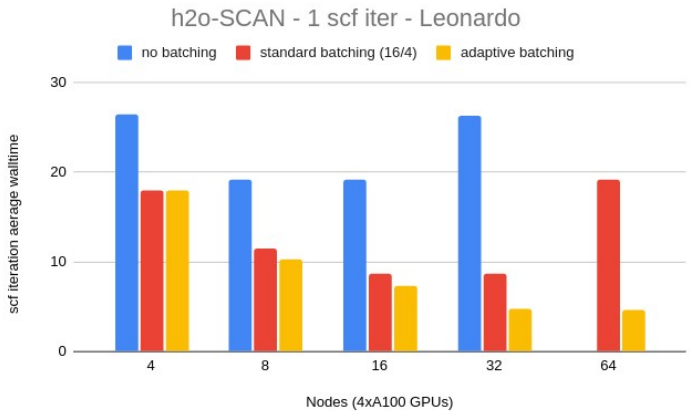

Figure 2: Scaling of a water system of 471 molecules with SCAN meta-GGA XC functional run with QUANTUM ESPRESSO on Leonardo cluster (CINECA). Blue bars represent calculation with no FFT batching in meta-GGA potential routines and band-sum ones, red bars represent the same water calculation with standard batching introduced where it was lacking and yellow bars represent, again, the same calculation with adaptive batching.

using the SCAN meta-GGA XC functional, run with PWscf on the Leonardo cluster (CINECA). As shown in Fig. 2, the absence of FFT batching in both the band-sum and meta-GGA potential routines results in poor scalability, with performance degradation evident as early as 8 nodes (blue bars). Introducing the standard batching scheme in these routines (red bars) leads to a marked performance improvement, sustaining scalability up to 16 nodes. The adaptive batching scheme (yellow bars), however, delivers the most significant gains, enabling efficient scaling up to 32 nodes and mitigating the communication overhead that would otherwise become critical above 32 nodes. A preliminary version of QUANTUM ESPRESSO with the extended batching can be found in a public `GitLab` repo. [5]

2.4 New scientific features: excited states' gradients

We have completed the first step in developing a NAMD workflow by implementing TD-DFPT transition energy gradients. This represents a significant advancement for the QUANTUM ESPRESSO suite, requiring extensive refactoring of both the `turbo_davidson` and `phonon` codes.

The gradients were derived from complete TD-DFPT equations while preserving the option to employ the Tamm-Dankoff approximation as needed. We adapted the Z-vector scheme—originally formulated for CIS gradients [1] and successfully implemented in other computational packages [2, 3]—to work within QUANTUM ESPRESSO 's plane-wave framework.

Our Z-vector solver employs a conjugate gradient approach to solve a single set of Sternheimer (CPHF) equations rather than 3N separate equations. We built this solver

[5] https://gitlab.com/fabrizio22/q-e/-/tree/more_fft_batching?ref_type=heads

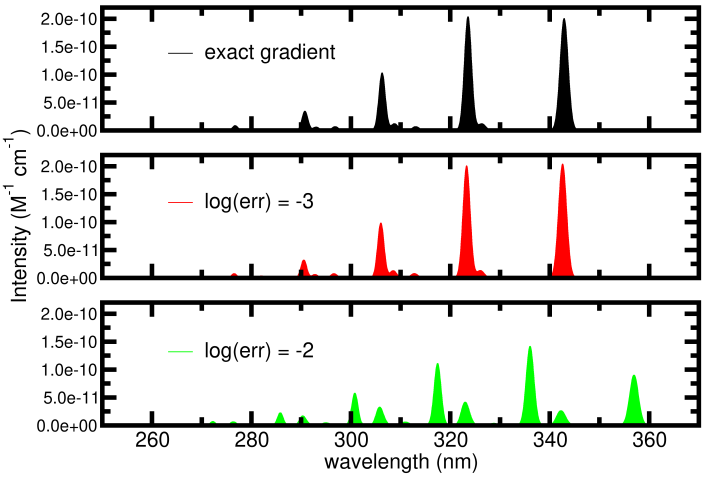

Figure 3: **Vibronic spectrum of gas-phase Formaldehyde** for the $S_0 \rightarrow S_1$ ($n \rightarrow \pi^*$, HOMO→LUMO) transition, computed using QE at the PBE/ONCV/60Ry level via the vertical gradient method in `FCClasses3` [4, 5]. Middle (red) and bottom (green) panels show spectral changes from added TD-DFPT gradient noise of 10^{-3} and 10^{-2} Ry/Bohr, respectively.

upon the existing implementation in the PHonon code at Gamma (`phcg.x`), which necessitated substantial code refactoring to relocate key routines into modules shared between TD-DFPT and PHonon. This approach enhanced code reusability while minimizing duplication.

We validated the gradient implementation through direct comparison of numerical and analytical force components for the first excited state of ten small molecules: ethene, acetylene, benzene, water, formaldehyde, acetaldehyde, acetone, ethanol, acetamide, and acetic acid. The average deviation was on the order of 10^{-4} Ry/Bohr, which we consider acceptable for all practical applications, as demonstrated by the sensitivity analysis presented in Fig. 3.

In this figure, the vibronic spectra of the formaldehyde molecule are computed using the vertical gradient approach, implemented in the `FCClasses3` code [4, 5]. Notably, significant changes in the excited-state properties are observed only when the gradient error exceeds 10^{-2} Ry/Bohr. Development is ongoing to finalise the TD-DFPT gradient implementation for hybrid functionals, as well as to enable GPU acceleration of the code. Additionally, a new workflow is being prepared to support excited-state geometry optimisations.

3 SIESTA

The second code release for SIESTA contains a number of new features, as well as performance enhancements and ancillary items to support deployment. Except when noted otherwise below, the material is included in the Gitlab release 5.4.0, [6] and in the repository for building and deployment support.[7]

3.1 New features

- We have completed the re-factoring and generalization of the QM/MM interface. This is a major milestone which will help addressing large problems, particularly in the area of electrochemistry, where liquid electrolytes that can be described in MM mode play a large role (see Fig. 4).

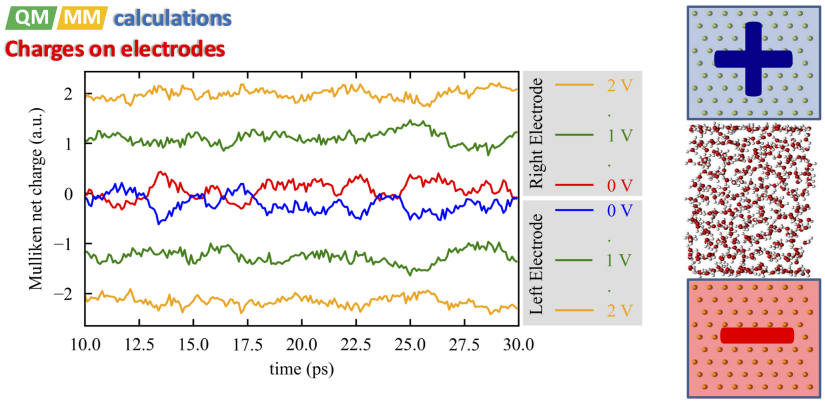

Figure 4: Sketch of the simulation cell for a biased system with gold electrodes described within QM and water molecules described within the MM scheme. Net charges on the electrodes during molecular-dynamics runs at different biases are shown in the left panel.

- We have updated the real-time TD-DFT subsystem in SIESTA, taking advantage of two separate strands of recent development: First, a new integrator option adds the time derivative of the overlap matrix to account for the moving basis rather than transforming to an orthonormal basis as with the currently implemented integrator. This new integrator (described in Ref. [6], Eq. 35) produces much better results for the electronic stopping power at high velocities, although it is not strictly unitary as the original integrator. Second, there is now a more efficient two-step propagation scheme which supports a larger time step and better energy conservation for extreme cases like inner-shell ionizations. This feature has not yet been merged into the stable community version of SIESTA, but is available as an advanced merge request. [8]

- A new utility computes the Lindhard function, which is particularly useful to analyze the stability and possible distortions of crystal structures, in particular in materials with effectively two-dimensional electronic structures.

[6] https://gitlab.com/siesta-project/siesta/-/releases/5.4.0
[7] https://gitlab.com/siesta-project/ecosystem/build-tools.git
[8] https://gitlab.com/siesta-project/siesta/-/merge_requests/470

- The implementation of spin-orbit capabilities in TranSiesta has advanced significantly, and has been used in production calculations [7], but the code and the new associated building system features are still in need of more thorough testing before they can be integrated into the main version.

3.2 Performance improvements and deployment

- In node-level performance, we have integrated support for newer versions of the ELPA library that offer GPU offloading of the Cholesky-decomposition step in the diagonalization workflow, and improvements and bug fixes related to the efficiency of other operations. In particular, new wrapping code and heuristics for the computation of the density-matrix from the eigenvectors have been added to our fork of the ELSI library, allowing it to offload this operation properly when using external ELPA libraries.

- The extra parallelization levels of TranSiesta are implemented, using appropriate library interfaces for direct inversion and the BTD (block tri-diagonal) method, but the integration and testing of the different components has not yet been completed. This development will allow the treatment of even larger systems in full QM mode, to complement the above-mentioned QM/MM capability.

- We have achieved initial deployment of SIESTA in all EuroHPC machines and performed appropriate benchmarks, using an updated (JUBE) framework developed in collaboration with WP3. It should be stressed that proper deployment and benchmarking have required a significant effort in discovering and implementing the appropriate low-level pinning instructions for each machine. This is an area which could benefit from clearer and readily available documentation in the EuroHPC realm.

- Deployment has been made easier by using the robust CMake foundation already reported on in the D1.2 deliverable, and by the development and refinement of complete Spack recipes for SIESTA itself and all its direct library dependencies, including the ELPA library. Supercomputing centers providing system-level Spack integration can use the recipes to create modules for users and for further development, benchmarking, and profiling. Even though Spack has been our HPC package-manager of choice, we have also adapted to EuroHPC guidelines by developing EasyBuild recipes, which are preferred in some machines such as LUMI, Karolina, and VEGA. Further, we have completed the integration of a recent version of SIESTA (5.2.2) in the EESSI framework for automatic deployment of HPC binaries (https://www.eessi.io/).

4 FLEUR

The second major release of FLEUR within this project will be the MaX-R8.0 release, as available at: https://iffgit.fz-juelich.de/fleur/fleur. Its most notable changes and developments are listed below.

4.1 Phonons in Density Functional Perturbation Theory

The implementation of phonons in FLEUR, reported on previously, develops into a valuable feature for understanding material properties. Here we report on this functionality and its state in the current FLEUR release, focusing on recent advancements, challenges, and performance enhancements.

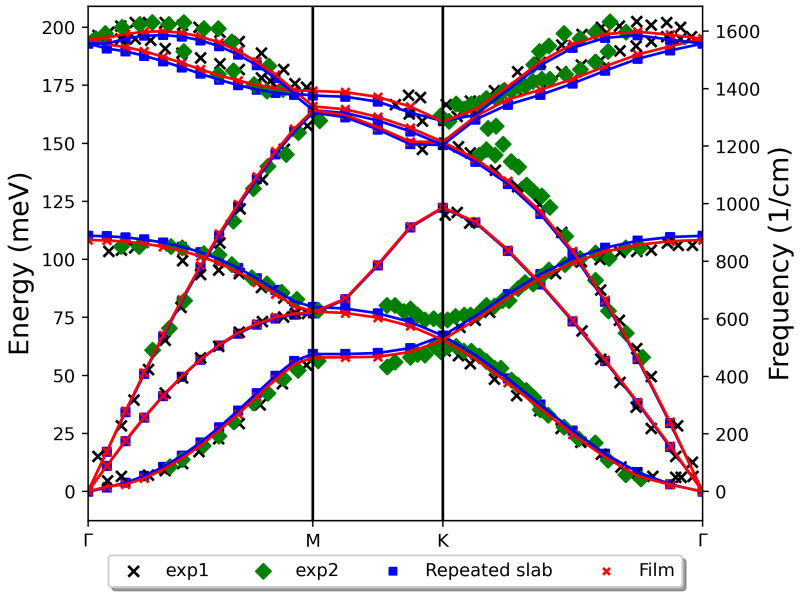

Figure 5: Phonon dispersion of graphene calculated with the bulk and film mode of FLEUR together with related experimental data. Note the good agreement of the different calculations, while the film mode is significantly more efficient. References for experimental data: exp1: H. J. Kröger et al., Small Methods **4**, 1900817 (2020), exp2: H. Yanagisawa et al., Surface and Interface Analysis **37**, 133 (2005).

For some time, FLEUR has utilised DFPT for calculating phonons. This method has been parallelised on multiple levels, with the most relevant being the parallelisation over k-points using the Message Passing Interface (MPI). This parallelisation enables efficient calculations of phonon frequencies and eigenvectors for bulk materials. To further enhance performance, the phonon implementation in FLEUR can utilise GPU-accelerated parts of basic FLEUR subroutines using OpenACC. Currently, the phonon code is thus capable of running on CPUs and GPUs with reasonable performance for small systems. In larger systems, further optimisations are necessary to ensure efficient calculations.

In this release, the phonon implementation in FLEUR has been extended to include the film mode, enabling the efficient simulation of 2D materials (Figure 5). This development is particularly significant, as 2D materials exhibit unique properties that are not

found in their bulk counterparts. The film mode enables the simulation of surfaces and 2D materials more efficiently, allowing researchers to explore the phonon properties of these materials with greater accuracy and speed. Without an explicit mode for two-dimensional structures, Phonons are often calculated in a so-called repeated slab calculation, which induces significant computational overhead and hard-to-control interactions between the slabs. This is avoided with our film mode.

However, the film mode also introduces new challenges:

- The film mode introduces another basis representation of the quantities in the vacuum regions. This required a corresponding extension of the FLEUR code to handle these new representations.
- The basis set in the film mode in the interstitial region has another interface compared to the bulk mode. This necessitates a careful treatment of the basis functions and their interactions with the vacuum regions.
- Numerical instabilities can occur due to this different basis set. These instabilities have been identified and resolved through careful analysis and optimization, ensuring the accuracy and reliability of the phonon calculations in the film mode.

A significant achievement of the phonon implementation in FLEUR is its performance in the film mode. Compared to the bulk mode in a supercell setup, the film mode is substantially faster, with a speedup of approximately 10x compared to repeated slab setups with distances frequently used. This performance enhancement is particularly important in the context of future automated workflows and high-throughput calculations.

4.2 New interface to linear algebra solvers

The FLEUR code uses different external libraries to perform linear algebra operations, most relevant being the solution of the generalized eigenvalue problem. In detail, we use libraries like LAPACK for operations on a single CPU, the ELPA, ChASE, SCALAPACK libraries for scaling to multiple nodes, i.e. in cases in which we have a distributed matrix. Similarly, for architectures using GPUs, we use CUBLAS and Magma on single nodes, while again ELPA and Chase can provide solutions for distributed problems. The key challenge from the point of view of the FLEUR code is to use a unified interface to these solvers to clearly encapsulate the functionality and to handle the different setup and input requirements of the different libraries. At the same time, the user should be enabled to control which solver is used to be able to select the most suitable algorithm for the problem at hand.

To achieve these goals, an extendable Fortran data type has been created that exposes the different capabilities of the solvers. In particular, the following methods can be provided:

- solve_GEV: Solve a generalized eigenvalue problem of the form $Hx = \lambda Sx$.
- solve_STD: Solve a standard eigenvalue problem of the form $Hx = \lambda x$.
- transform: Transform the generalized eigenvalue problem to a standard eigenvalue problem, e.g., by performing a Cholesky factorization of the matrix S.

- backtransform: Transform the solution of the standard eigenvalue problem back to the generalized eigenvalue problem.
- solve_STD_SP: Solve a standard eigenvalue problem of the form $Hx = \lambda x$ in single precision.

The interface is designed to be extendable, so that new solvers can be added in the future. Beyond the automatic selection of default solvers, the users now have the option to select the steps of the eigensolver procedure at runtime by providing command line options or by setting the corresponding environment variables in the input file.

4.3 First tests in single precision arithmetics

Using the new interface to the eigensolvers, we tested the possibility to employ mixed precision calculations in the FLEUR code. As our overlap matrix is nearly singular, solving the generalized eigenvalue problem is very sensitive to numerical precision and the use of single precision does not yield satisfying results. However, we found that a mixed approach of using double precision for the reduction to the standard eigenvalue problem and the back transformation to the generalized eigenvalue problem, combined with single precision for solving the standard eigenvalue problem yields excellent agreement with the reference double precision results (Figure 6).

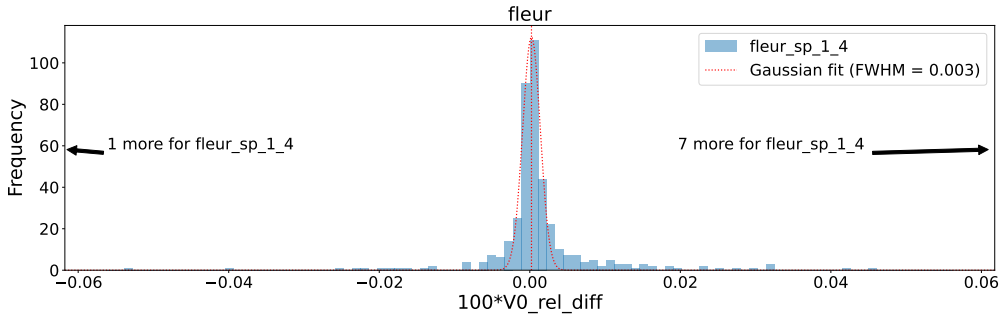

Figure 6: Comparison of the results of the FLEUR code using single precision for solving the standard eigenvalue problem compared to the double precision results. Shown are the frequencies of the differences in the equilibrium volume for an extensive set of test cases in a histogram plot. The Gaussian fit with a full width at half maximum of 0.003% gives a scale for expected deviations between the two approaches.

4.4 Adjustment of FLEUR to run on JEDI

In preparation for the upcoming JUPITER system, the FLEUR code has been adjusted and tuned. To achieve this, we participated in the JUPITER Research and Early Access Program (JUREAP) in which we prepared the code to run as large-scale jobs on the JUPITER architecture. This involved two basic tasks:

- The porting of FLEUR to the architecture of JUPITER.

- The testing of the code as large-scale jobs on the full JUWELS booster partition to demonstrate the scalability of the code and to ensure that large-scale jobs run correctly.

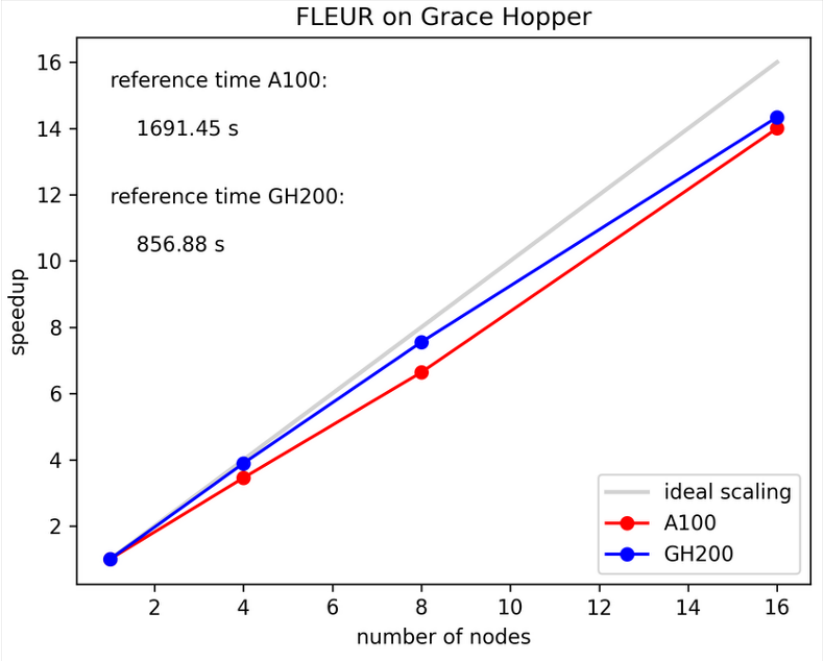

Figure 7: Scalability of FLEUR on JEDI compared to the A100 GPUs provided by JURECA-DC. All simulations were done with one MPI task per GPU (4 per node).

The porting of FLEUR to the architecture of JUPITER was performed on the JEDI system, which is a smaller version of JUPITER. The JEDI system is based on the same architecture as JUPITER, but has fewer nodes. The porting process involved adapting the code to the specific hardware and software environment of JEDI, including the use of the required libraries and the specific compiler settings for the JUPITER architecture. The successful execution of benchmark tests ensured that the code runs correctly and efficiently (Figure 7). The build process was made robust and reproducible using the Spack package manager with customized environments for the software stack on JEDI which is supposed to be transferred to JUPITER.

The code was tested on large-scale jobs using the full JUWELS booster partition with different workloads and configurations (Figure 8). In this process some hardware issues on JUWELS were identified which delayed some of our benchmarking runs significantly. At the end of the testing phase, we were able to demonstrate that FLEUR can run efficiently on the JUPITER architecture and that it is capable of handling large-scale jobs. A corresponding certificate was issued by the JUREAP team (Figure 9) and the FLEUR team was able to ensure early access to the JUPITER system for further testing and development.

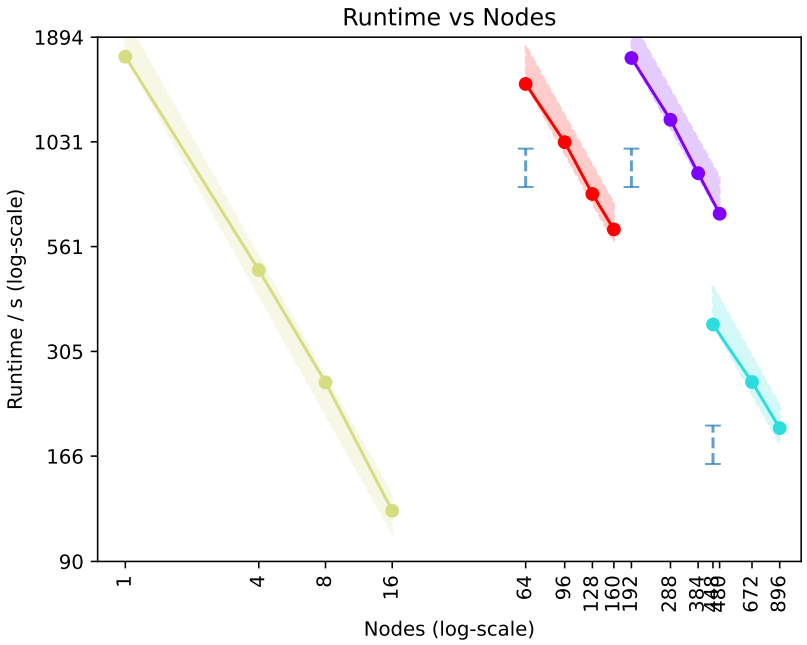

Figure 8: Combined plot demonstrating the scalability of FLEUR on JUWELS booster for different workloads.

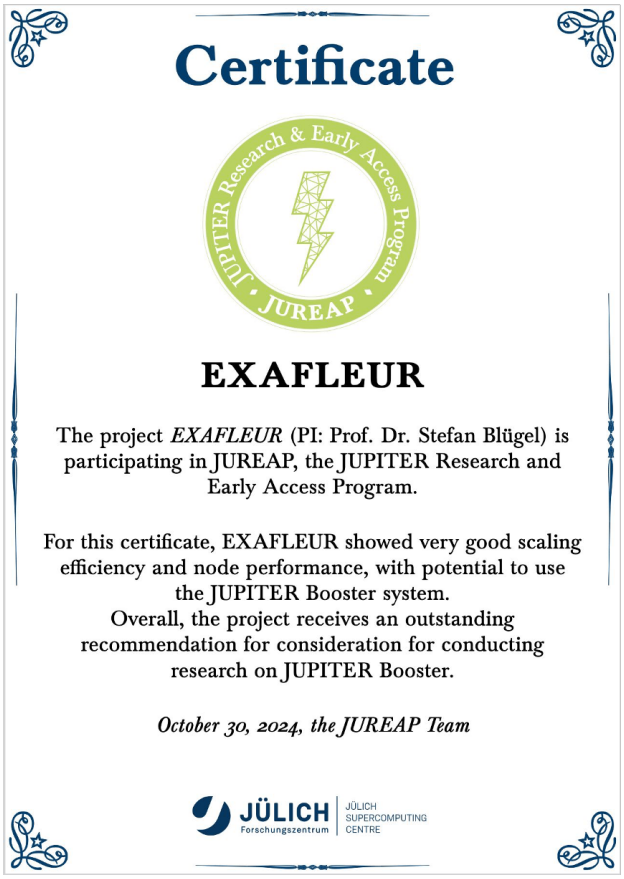

Figure 9: JUREAP certificate on FLEUR's readiness to run on JUPITER.

5 BIGDFT

5.1 Node-level Performance

One of the critical aspects of preparing MAX lighthouse codes for exascale is achieving efficient and portable node-level performance, especially for heterogeneous architectures equipped with modern accelerators. In this context, the BigDFT code has made substantial progress in establishing a clean and flexible device offloading infrastructure. This work focuses particularly on the GPU-accelerated Poisson solver, which is a key computational kernel within the code.

Over the past months, we have developed and validated a new SYCL-based interface to the Poisson Solver, designed to support multiple FFT backends through a unified abstraction layer. This interface, inspired by a so-called *Hourglass architecture*, allows the runtime selection of available backends—such as CUFFT, MKL, rocFFT, and DBFFT—without requiring changes in the core algorithmic logic. This design enables code portability and performance tuning across a wide range of hardware platforms.

The ONEMath library was introduced as a selector layer implemented in SYCL/C++, responsible for encapsulating FFT backend functionality and providing seamless interoperability with existing Poisson solver logic. Its design ensures minimal overhead while retaining the flexibility required to adapt to platform-specific optimizations. Notably, the library supports:

- **CUFFT:** NVIDIA's highly optimized FFT implementation, used in existing CUDA-based BigDFT kernels.
- **MKL-DFT:** Intel's FFT solution for CPU and GPU acceleration, integrated through oneAPI.
- **rocFFT:** AMD's GPU acceleration library, allowing deployment on systems with Instinct accelerators.
- **DBFFT:** An OpenCL-based FFT library specifically written to serve BigDFT. This backend is especially important for portability, as it does not require the Intel Graphics Compiler (IGC), making it suitable for ARM-based and other OpenCL-compliant platforms.

This refactoring is more than a backend switch structure: it represents a foundational step toward creating a single-source offloading strategy based on SYCL and C++ templates, aligning with modern HPC trends. It also reduces the burden on developers and facilitates a maintainable, performance-portable codebase.

Functional validation of the new backend infrastructure has been completed successfully across a range of platforms, including Intel's Ponte Vecchio (PVC), AMD MI-series GPUs, and NVIDIA A100 GPUs. Performance benchmarks and scaling analysis are ongoing and will be further enhanced with the integration of a Kokkos-based acceleration path. This planned addition will allow a more fine-grained optimization of memory access patterns and parallelism across execution spaces (e.g., CPU, CUDA, HIP, SYCL), while maintaining a common computational kernel interface.

The developments achieved in this task demonstrate the feasibility of a truly portable and multi-backend numerical core within a complex electronic structure code like BigDFT.

This work lays the groundwork for future exascale deployment, in which hardware diversity and energy efficiency will be essential.

5.2 Parallel Efficiency

Efficient parallel scalability is central, and for the BigDFT code, this challenge is particularly relevant in its linear-scaling (LS) mode, which is designed to treat very large systems with thousands of atoms. In the context of the MAX project, significant attention has been devoted to the stability and scalability of the MPI communication infrastructure, especially for systems with a large number of MPI ranks and diverse interconnect architectures. Early in the project, some sporadic runtime instabilities were observed when running LS-BigDFT on specific clusters, notably those relying on MPICH-based MPI libraries. Although the one-sided MPI communication paradigm used in BigDFT had generally proven to be efficient and robust, a detailed investigation was launched to understand these anomalies. This investigation led to the discovery of a subtle but critical bug in the communication of the density matrix.

Specifically, the LS-BigDFT kernel (notably in `rhopotential.f90`) was found to invoke a native MPI call (`mpi_get`) with a `futile`-wrapped window object instead of its underlying MPI handle. This mismatch, while going unnoticed by the compiler due to pointer aliasing, caused undefined behaviour on some platforms. The fix involved correctly replacing the native MPI call with the appropriate wrapper from the `futile` library, ensuring compatibility between the call and the window abstraction.

Beyond this major fix, several other improvements were implemented to enhance the stability and load balancing of MPI communications:

- The optional algorithm replacing `mpi_alltoallv` with multiple `mpi_get` calls, previously enabled by default, has now been restricted to only activate when explicitly triggered via environment variables. This change improves robustness on systems where the replacement approach performs poorly.
- In some scenarios, zero-sized MPI windows were being created, which, although standard-compliant, led to instabilities. The `futile` wrapper has now been hardened to explicitly manage such cases.
- Blocking reductions (e.g., `mpi_allreduce`) used in small-scalar communications, such as Hamiltonian traces, have been replaced with non-blocking collectives (`MPI_IALLREDUCE`) where applicable, improving asynchronous execution and reducing communication stalls.

While no single modification was solely responsible for resolving the encountered issues, the combined set of changes has significantly improved the robustness and scalability of LS-BigDFT. Benchmarks on large biomolecular systems now show stable runs across a wider range of node counts and machine configurations, including those with high network contention or weaker collectives. These improvements are now integrated into the latest public release of BigDFT and contribute directly to the exascale readiness of the code. They also serve as a blueprint for future developments focused on advanced load balancing strategies and hybrid MPI + task-based parallelism.

5.3 Software Engineering

Maintaining a modern and portable electronic structure code requires a robust software engineering infrastructure that can accommodate a wide variety of backends, architectures, and evolving programming models. In the framework of the MAX project, the software engineering efforts for `BigDFT` have primarily focused on enhancing modularity, portability, and code maintainability — all critical for ensuring long-term sustainability and ease of deployment on pre-exascale and exascale platforms.

A key development in this context has been the implementation of a unified abstraction layer for the BigDFT FFT-based Poisson solver. The new interface, written in SYCL and C++, enables seamless switching between various FFT backends, including Intel's `MKL-DNN`, NVIDIA's `cuFFT`, AMD's `rocFFT`, and the OpenCL-based `DBFFT` library developed in-house. This SYCL-based abstraction, integrated through a component called `ONEMath`, introduces an Hourglass-like interface architecture. Developers now write backend-agnostic Poisson solver code, while users can select the desired backend at runtime or compile-time.

The `DBFFT` backend deserves specific mention: it is not a legacy implementation but rather a dedicated OpenCL FFT library designed for portability. It operates efficiently on ARM-based systems and other platforms where Intel's graphics compiler framework is unavailable. The library has shown good performance on Intel's PVC accelerators and is compatible with a variety of Linux-based systems, further enhancing `BigDFT`'s architectural coverage. These developments substantially reduce code complexity and facilitate integration into evolving compiler stacks such as Intel's DPC++ (Data Parallel C++) and Codeplay's Open SYCL. The benefits are twofold: (i) the same core implementation can be deployed across multiple accelerator architectures, and (ii) the development team can minimize architecture-specific branching within the main codebase.

To support future extensions and performance tuning, work is ongoing to integrate the `Kokkos` programming model. Kokkos will provide a portable backend for structured grid computations and kernel launches, particularly beneficial for the Poisson solver and real-space operator applications in `BigDFT`. This effort complements the SYCL work and is designed to explore hybrid pathways for cross-architecture performance portability. Finally, these software engineering improvements are being incorporated into the `futile` utility library, which acts as the abstraction layer for many of the internal services used by `BigDFT`, including I/O, MPI wrappers, and now, kernel execution. All developments adhere to modern C++ standards and are continuously tested and validated through CI pipelines to ensure stability across different target platforms.

These enhancements position `BigDFT` as a forward-compatible codebase, capable of evolving with the high-performance computing ecosystem while remaining accessible to its developer and user communities.

5.4 Interoperability Example: Dynamical QM/MM Workflow for Drug Resistance Prediction

This work represents a continuation and refinement of the methodology established in the project's first deliverable, which focused on the design of multi-scale simulation tools to couple quantum and classical approaches. In the present phase, we extend that foundation by incorporating *dynamical effects* into the quantum-classical framework. This

is achieved through the integration of GPU-accelerated molecular dynamics simulations with linear-scaling quantum mechanical calculations, coordinated via a fully automated, sample-driven workflow. The orchestration is performed by the `remotemanager` Python framework, which enables asynchronous, event-driven control of coupled QM/MM simulations. This advancement allows the system to efficiently sample relevant conformational states while performing targeted quantum analyses on selected configurations, offering a more comprehensive picture of molecular recognition processes and drug resistance mechanisms.

Background. One of the challenges of developing medicines for viruses is drug resistance. Though a proposed drug may effectively inhibit key viral proteins, such inhibition does not last as evolutionary pressure leads to new variants. Thus, essential medicines like Darunavir, used to treat HIV/AIDS, are at risk of becoming ineffective. It has been recently pointed out that modification of the structure of systems far from the active site (so-called "distal mutations") can impact drug effectiveness. Sophisticated methods that reveal information about the entire protease-drug complex in its many conformations are necessary [8].

Recently, we have been implementing a framework connecting the BigDFT code, able to treat systems of the size of HIV-1 protease (3,000–4,000 atoms) robustly and efficiently [9], with the Genesis classical molecular dynamics (MD) code, which is optimized for GPU architectures [10]. By combining these codes, we aim to realize the in-depth electronic structure picture provided by DFT calculations [11], while suitably sampling the conformer space using a combined quantum mechanics/molecular mechanics (QM + MM) workflow [12], not to be confused with the more traditional QM/MM approach.

Methods. To implement this approach, we deployed both BigDFT and Genesis on two different supercomputers, leveraging its two distinct partitions: a CPU-optimized partition used for the BigDFT calculations, and a GPU-accelerated partition used for the Genesis molecular dynamics runs. A custom workflow was built using our `remotemanager` Python library [13], allowing both codes to operate in a coupled yet asynchronous fashion, which is orchestrated from the local machine of the user.

The Genesis MD simulation runs continuously on GPU nodes, producing molecular snapshots over the course of several days. Meanwhile, a remotemanager-driven process monitors the output directory, detecting new snapshots as they are generated. Upon detection, it immediately submits a corresponding BigDFT DFT calculation on the CPU partition. This "light daemon" likewise detects completed BigDFT runs, triggering post-processing routines to extract key quantum observables relevant to the study of the protease-drug binding network. This streaming approach ensures low time-to-solution, as quantum analyses are performed concurrently with classical MD sampling.

In practice, we found that the Genesis calculations were optimally performed using a single node of the GPU-accelerated partition, while each BigDFT calculation was assigned 64 nodes of the CPU-optimized partition (based on A64fx architecture in our tests). Post-processing tasks were executed efficiently on a single node. This infrastructure allowed for high-throughput quantum sampling from long MD trajectories.

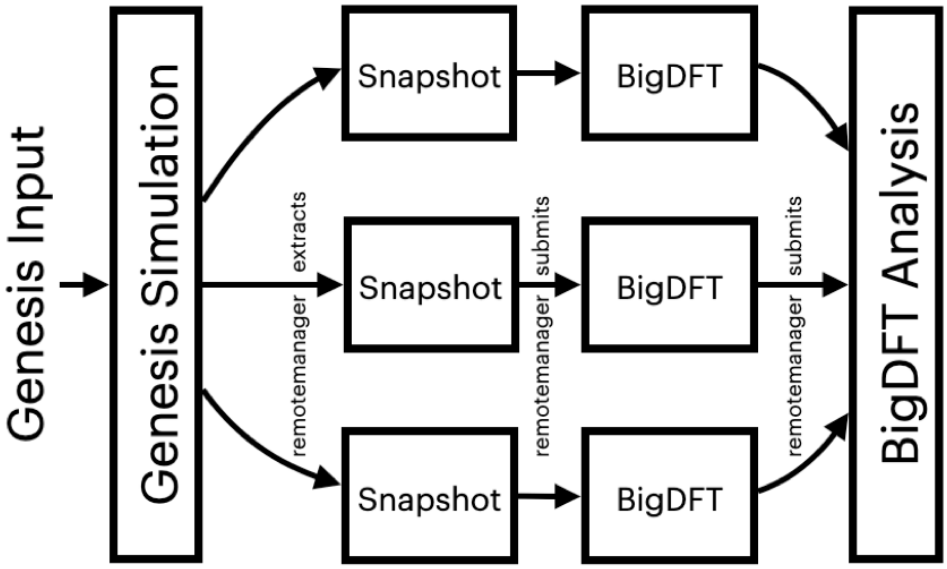

Figure 10: Combined BigDFT and Genesis workflow (left) used for studying Darunavir bound to HIV-1 protease (right). Both codes simulate systems with explicit water; BigDFT considers only water within 3.0 Å of the complex.

The workflow was applied to three experimentally characterized variants of HIV-1 protease bound to Darunavir: the wild type (PDB: 6OPS), a double-mutant variant (6OPT), and an 11-mutation variant (6OPZ). These systems were simulated in the presence of explicit water molecules; however, BigDFT calculations were restricted to water within a 3.0 Å cutoff from any atom in the protein-ligand complex to maintain tractability.

Results. In Fig. 11, we show the interaction networks between Darunavir and the three HIV-1 protease variants. When we compute the total interaction energy between Darunavir and the protease alone (excluding water interactions), our results qualitatively reproduce trends observed experimentally. Notably, the interaction networks associated with distal mutations exhibit increased numbers of water-mediated contacts with the drug molecule.

This suggests that, beyond the expected effects of binding-site mutations, distal mutations increase the solvent-accessible surface area of the ligand, weakening the overall drug-protein binding and potentially contributing to reduced inhibitory efficacy.

Conclusions and Outlook. Our workflow demonstrates the feasibility and value of combining high-throughput, GPU-accelerated MD sampling with linear-scaling QM calculations to explore drug resistance mechanisms in realistic, solvated systems. Future work will expand the analysis to cover a broader set of HIV-1 variants and longer MD trajectories. This prototype approach will potentially enable a statistical treatment of interaction variability and support the identification of design strategies for next-generation inhibitors.

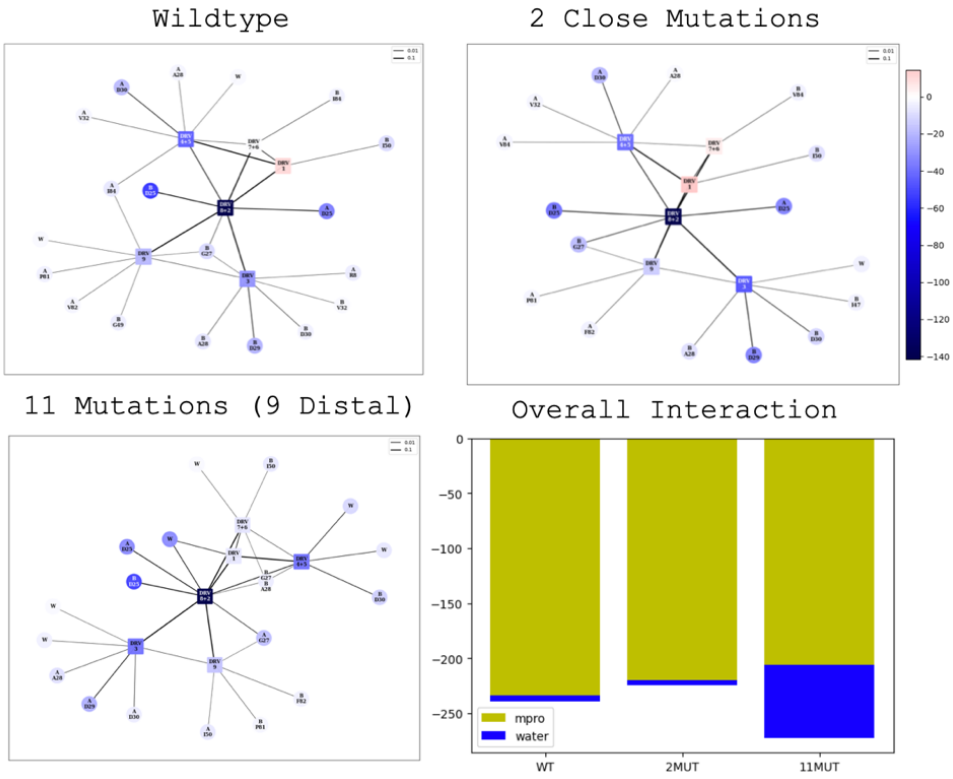

Figure 11: Interaction graphs of Darunavir with different variants of HIV-1 protease. A weakening of binding interactions is observed as the number of mutations increases.

6 YAMBO

The M30 software release of the YAMBO code consists of several performance improvements, including HPC optimizations, algorithmic refactoring, improved GPU porting, better exploitation of external tools such as linear algebra kernels. A number of scientific features and capabilities have also been added to the code. In terms of software engineering, Spack recipes have been consolidated, while some preliminary EasyBuild recipes have been released. The portfolio of EuroHPC machines where the code have been deployed has been enlarged and more benchmarking data collected. Unless differently stated, the above mentioned developments are available on the official GitHub repo of YAMBO,[9] in the `tech-master`, `tech-gpu`, and `tech-optimiz` branches. These are the basis for the next stable community release of the code, YAMBO v5.4.0.

6.1 Performance improvements

DeviceXlib developments. The OpenACC and OpenMP-GPU support of DeviceXlib has been stabilized and consolidated. In particular, a more consistent handling of OpenACC queues and interaction with CUDA streams has been implemented. Some interfaces had been updated, wiht the idea of supporting a logic where, in the absence of GPU devices, a proper CPU-only fallback is in place. Support of `int64` integers added, as needed, e.g. by `cusolverMP`. Support of ARM and Fujitsu (incl. Fugaku and Deucalion) architectures added. A new tag (v0.9.0) of devxlib has been created, available on GitLab.[10]

Response function optimization. One of the key kernels controlling the overall performance of the YAMBO code is the calculation of the irreducible response function χ_0. In view of this, together with WP4 (Codesign), the YAMBO team extracted a mini-app[11] dedicated to the response function. This was analyzed and profiled by Eviden, CINECA, and IT4I personnel, who identified some performance critical sections (notably, the filling of the response function using triangular matrices by means of BLAS-1 operations). This led to a set of recommendations, that were taken and implemented by the YAMBO team in the mini-app, resulting in rather large performance improvements (working with matrices up to 1000×1000, speedup larger than 2x were obtained, both on CPUs with OpenMP and on GPUs).

Next, the optimization blueprint has been transferred from the mini-app to the whole YAMBO. This took a relatively short time, leading to sensible performance improvements, as shown in the tables below, with improvement in the time-to-solution of χ_0 up to -40% (then resulting in a similar improvement in the global time-to-solution for the benchmarks considered, since dominated by χ_0).

Two additional improvements originate as byproducts of the operation: (i) removing the handling of the triangular representation of the residues in the response function led to a simplification and streamlining of the code, with also a reduction of the memory footprint; (ii) The BLAS-1 operations removed were replaced by GEMM operations, which are now placed in one of the performance cores of the code. This is very interesting

[9] https://github.com/yambo-code/yambo
[10] https://gitlab.com/max-centre/components/devicexlib/-/tags/0.9.0
[11] https://gitlab.com/max-centre/components/mini-apps/-/tree/main/Yambo

also in view of exploiting reduced precision hardware with FP64 emulation (according e.g. to the to-called Ozaki scheme [14]).

Table 1: Performance comparison of YAMBO between version 5.3.0 and the branch with the response function optimization on both Leonardo partitions.

Leonardo-Booster — grco-small (7k) benchmark					
Yambo Version	Nodes	Tasks	Threads/Task	Time-Profile [s]	Xo [s]
5.3.0	4	16	8	597.00	504.23
tech-optimiz	4	16	8	377.00	282.22
Speedup				*1.58*×	*1.79*×
Leonardo-Booster — grco-large (61k) benchmark					
Yambo Version	Nodes	Tasks	Threads/Task	Time-Profile [s]	Xo [s]
5.3.0	80	320	8	5400.00	4980.00
tech-optimiz	80	320	8	3449.00	3024.00
Speedup				*1.57*×	*1.65*×
Leonardo-DCGP — grco-small (7k) benchmark					
Yambo Version	Nodes	Tasks	Threads/Task	Time-Profile [s]	Xo [s]
5.3.0	4	32	14	3508.00	2902.00
tech-optimiz	4	32	14	1971.00	1340.00
Speedup				*1.78*×	*2.17*×

Interface with cuSolverMP. The main runlevels of YAMBO, notably including GW and BSE, make use of dense linear algebra (LA) operations, typically in the form of linear system solvers (as in the Dyson equation involving the reducible and irreducible response functions), matrix multiplication (as in the evaluation of the irreducible response), or eigenvalue solvers (as in the BSE). Depending on the use-case, these problems may involve dense matrices large enough to be critical in terms of time-to-solution and memory-footprint. In these cases, the use of distributed LA kernels is advisable. While Scalapack LA was introduced in YAMBO a few years ago, distributed linear algebra on GPUs was not available so far.

During the present reporting period, a team led by CINECA personnel has implemented in YAMBO the support for NVIDIA `cusolverMP`, initially for what concerns the linear system problems. Numerical results, as those reported in Fig. 12 (obtained with a χ_0 matrix of size $20k \times 20k$) show that `cusolverMP` is able to strongly reduce the time-to-solution for the LA kernel as compared to `ScaLapack`. Further optimization of the interface is still ongoing, together with the extension to Hermitian eigenproblems (BSE).

Other GPU-oriented optimization. Concerning the GPU porting of YAMBO, work has been done along three main lines:

		cuSOLVERMp		ScaLAPACK		
Nodes	Tasks/Node	Linear Algebra	getrf/getrs	Linear Algebra	SU LA	SU func
1	4	118.16	73.58	14760.00	124.9x	200.6x
4	4	116.46	61.05	4080.00	35.0x	66.9x
16	4	119.72	60.41	1063.00	8.9x	17.6x

Figure 12: Comparison of `cusolverMP` and `ScaLapack` in solving a dense linear system of size 21453×21453, as required by the calculation of the response function χ in YAMBO.

- **Porting of features that were not available or optimized with GPU support.** Examples of these activities include the GPU porting of the GW MPA self-energy, or the porting of new features under development (such as the W-average approach for metals, not included in the present software release).
- **Optimization of the OpenACC porting in comparison with CUDA-Fortran on NVIDIA machines.** This is probably the activity that took the largest amount of effort. Based on the observation that, for the very same input files, YAMBO runs using CUDA-Fortran as backend for GPU support were much faster than OpenACC ones (with a 2x factor), together with CINECA and IT4I personnel we have investigated the issue. The YAMBO mini-app described in the above Section turned out very useful also in this piece of work. We used the mini-app for profiling and identified a few critical aspects in the YAMBO implementation of OpenACC support. Among these, the use of `sync` parallel loops and the decoration of each loop with a `data present` region. Properly handling these aspects, we were eventually able to significantly reduce the OpenACC time-to-solution in the mini-app to a value similar to that of CUDA-Fortran (say within a 10% in time-to-solution). These fixes were already ported to DeviceXlib and work is ongoing to port them also into YAMBO.
- **Consolidation of the OpenMP-offload porting of YAMBO on LUMI.** While a working porting on LUMI of YAMBO has been available since the end of 2023, during the RP2 reporting period we have consolidated the porting (based on OpenMP-offload), including small bug fixes, and, notably, the support of the Cray compiler v17. Benchmarking activities (as reported below) show that the YAMBO code runs GW workflows on LUMI with a per-node performance similar to that of the NVIDIA-based Leonardo Booster machine.

6.1.1 More on advanced interfaces with linear algebra solvers

As mentioned, advanced interfaces with linear algebra (LA) solvers are essential for tackling large-scale problems, such as those anticipated for MAX showcases. In the last year, we have extended the number of linear algebra libraries interfaced with YAMBO. The `Ydiago` library was created as an external library (public repository available under the yambo-code organization) [15], which works as a layer between YAMBO and both Scalapack and ELPA. Thanks to `Ydiago` we obtained a significant improvement in the extraction of eigenvalues and eigenvectors of the BSE matrix. Moreover, the library also

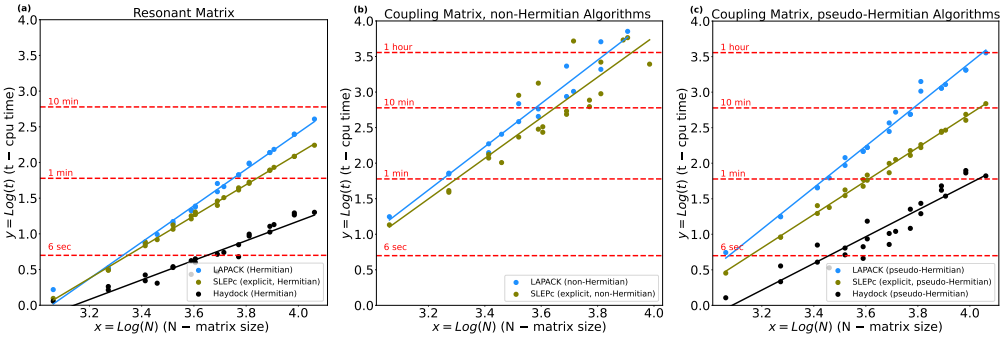

Figure 13: Time complexity plots on 1 CPU for solving the BSE eigenvalue problem in (a) the Hermitian case, (b) coupling case with a non-Hermitian algorithm (yambo 5.3 implementation), and (c) coupling case with a pseudo-Hermitian algorithm (new implementation). Simulations data for exact diagonalization (Lapack), iterative solution (Slepc), and.

works with ELPA compiled with GPU support, thus making it possible to solve the BSE matrix on GPU-accelerated HPC machines. Finally, the `Ydiago` library is also able to exploit the pseudo-Hermitian structure of the BSE matrix, as required when solving the Bethe-Salpeter equation beyond the Tamm-Dancoff approximation (see Fig. 13, blue data).

Similarly to the full diagonalization case, the implementation of iterative solvers via the `SLEPc` library was also significantly improved, thanks to a collaboration with the team of the `SLEPc` developers, started about one year ago (see D2.2). The project is now largely completed, with the iterative solver (i) now able to run on GPUs, and (ii) also able to take advantage of the pseudo-Hermitian structure of the BSE matrix (see Fig. 13, green data). Details on the algorithm and scaling performances are available in two preprints [16, 17].

6.2 Deployment & Benchmarks

Spack recipes further developed. Recently, the CINECA user support team prepared a new software stack for Leonardo-Booster. In this stack, updated versions of compilers and the CUDA toolkit were used to compile all libraries and applications. In particular, CUDA toolkit version 12.2 was chosen as the default, as it is considered the base version to ensure forward compatibility with the CUDA driver installed on the compute nodes.

The Spack recipes prepared for the YAMBO code and the `DeviceXlib` library were not able to compile with this version of the CUDA toolkit and the selected NVHPC SDK version (24.5). This issue was caused by a bug in the `configure` script of both YAMBO and `DeviceXlib`. For the library, we decided to fix the bug and create a new version tag (0.8.6). Regarding the YAMBO code, version 5.3.0 was released recently, and the fix, already implemented, will be included in the upcoming patch release 5.3.1. In the meantime, a patch has been added to the Spack recipe for YAMBO to apply the necessary fix directly to the `configure` script.

EasyBuild recipes developed. In the context of the work carried out by the CI/CD working group organized by CASTIEL2, we started developing EasyBuild recipes for the YAMBO code and the `DeviceXlib` library. A working recipe is now available using the `foss-2023a` toolchain, which already includes several libraries required by YAMBO.

Unfortunately, one of YAMBO's dependencies, the PETSc library, did not have an available recipe that supports compilation in single precision. To address this limitation, it was necessary to significantly modify the EasyBuild recipes for both PETSc and SLEPc. In passing, we note that one of the strategies currently explored to improve performance in HPC applications is reducing the floating-point precision while maintaining acceptable accuracy in the results. In EasyBuild, modifying the behaviour of an installation typically requires creating a new recipe with the necessary changes. In contrast, Spack allows for greater flexibility by defining variant, compilation options that can be enabled or disabled via command line arguments.

The newly developed EasyBuild recipes were successfully tested on the Deucalion supercomputer, on both `x86` and `arm` partitions. This confirms that the same set of recipes can be used to compile the software on both architectures using the `GCC` compiler suite. In addition, we also developed EasyBuild recipes for the LUMI-C supercomputer. On LUMI, custom toolchains have been defined, namely `cpeGNU` and `cpeCray`. We managed to create functional recipes for both toolchains. Performance testing revealed that the YAMBO code performs better when compiled with the `cpeGNU` toolchain.

ARM and MAC compilation. YAMBO has been ported to ARM architectures, from version 5.3.0, enabling its use on supercomputers like Fugaku and Deucalion (ARM partition). This allows the development and testing of new algorithms to improve parallel efficiency and support calculations of electronic and optical properties of complex materials. The long-term goal is to ensure readiness for deployment on JUPITER, the first European exascale supercomputer. Additionally, supporting installation on ARM-based workstations and laptops (e.g., Apple devices) improves energy efficiency and allows researchers to run tests without immediate access to HPC systems.

The initial porting to Apple devices was straightforward, requiring only minor adjustments during the configuration phase, such as recognizing the `aarch64-apple-darwin23` platform and selecting appropriate compilation flags. YAMBO was successfully installed using the GCC compiler suite, OpenMPI, and dependencies managed via Homebrew (more details on this technical report [18]). Subsequently, access to the Fugaku supercomputer allowed us to extend support by adding recognition for the `aarch64-unknown-linux-gnu` platform. Installation was facilitated by the existing Spack recipe. Finally, YAMBO was also compiled on the Deucalion supercomputer through the development of a dedicated EasyBuild recipe.

Conda-forge recipe. In order to make the YAMBO code easily installable on systems such as personal computers and workstations, thus facilitating its adoption and learning, a recipe has been developed to enable YAMBO installation via the `conda` package manager and the conda-forge project [12]. `conda` is a package manager and environment manager that works across the three main operating systems: Linux, macOS, and Windows.

[12] https://anaconda.org/conda-forge/yambo

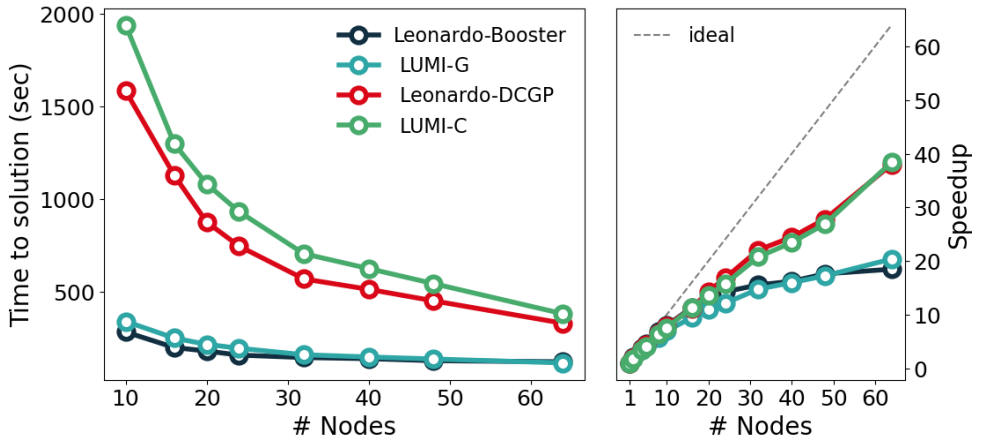

Figure 14: GrCo-small benchmark, comparison of strong scalability and speedup with YAMBO v5.3.0 on Leonardo at CINECA and LUMI at CSC.

Currently, the recipe allows for the installation of the latest YAMBO release (5.3.0) on Linux for the three architectures supported by conda-forge, x86, ARM, and PPC, and on macOS with the x86 architecture. As mentioned in the previous section, YAMBO can also be installed on macOS ARM, but the development of the conda-forge recipe is encountering difficulties due to the fact that the CI pipeline used by conda-forge emulates the ARM architecture starting from x86. While this is not an issue in the Linux environment, it proves to be more challenging to manage on macOS.

Benchmarks on EuroHPC pre-exascale machines. This paragraph presents the results of ongoing development work, with a particular focus on GPU porting, through periodic benchmarks and comparisons across different architectures. Figure 14 shows a comparison in terms of strong scalability and speed-up using the GrCo-small benchmark,[13] between the two supercomputers Leonardo and LUMI, for both accelerated and non-accelerated partitions. Figure 15, shows a comparison limited to the Leonardo-Booster and LUMI-G partitions using the GrCo-large benchmark.[14] An interesting aspect of the latter comparison is that the deployment work carried out on LUMI-G, following the introduction of the new software stack (LUMI/24.03, with Cray compiler v17), enabled performance levels comparable, though still slightly lower, to those of the more mature CUDA-Fortran porting.

Recently, YAMBO was also successfully deployed on the MareNostrum 5 supercomputer (ACC partition, with 4 Nvidia H100 GPU cards per node) by colleagues at BSC, using the Spack recipe. A dedicated module is now available for users. Additionally, they conducted a series of experiments using the GrCo-small benchmark and the JUBE workflow manager, testing various configurations for the number of processes per GPU, see Figure 16. The results revealed some interesting behaviors, particularly when MPS [15]

[13] https://gitlab.com/max-centre/benchmarks-max3/-/tree/master/yambo/grco-small-ppa
[14] https://gitlab.com/max-centre/benchmarks-max3/-/tree/master/yambo/grco-large-ppa
[15] https://docs.nvidia.com/deploy/mps/index.html

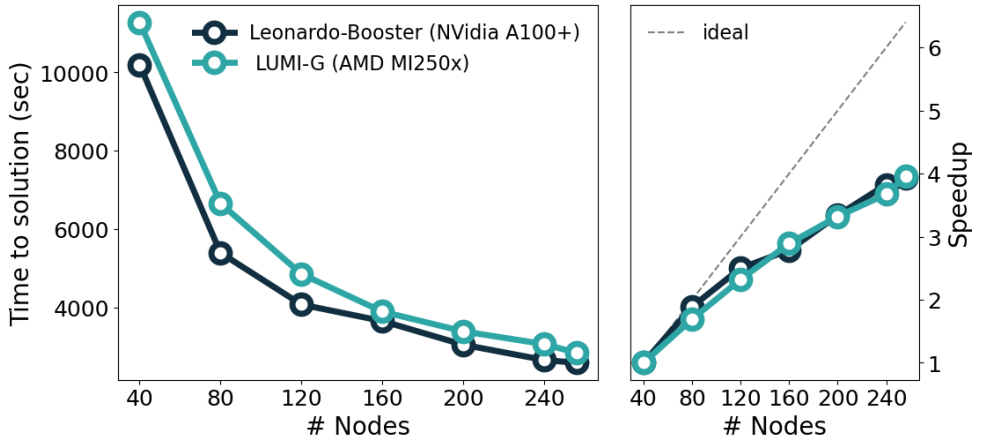

Figure 15: GrCo-large benchmark, comparison of strong scalability and speedup with YAMBO v5.3.0 on Leonardo at CINECA and LUMI at CSC.

(Multi-Process Service) was enabled and a smaller number of nodes was used. MPS allows multiple processes to share GPU resources concurrently, improving utilization and reducing latency. This is especially advantageous for workloads composed of many small tasks, or when running multiple jobs on the same GPU. However, using an excessive number of MPI processes with MPS can saturate GPU resources, causing contention, increased latency, and ultimately reduced performance.

6.3 New features and Interoperability

Spin flip response function within BSE and TDDFT. In the develop version of YAMBO the calculation of the response function χ_{+-} in the spin-flip channel has been coded. χ_{+-} gives access to magnon dispersion which can be described with both TDDFT and BSE (at variance with the spin conserving channel, where excitons physics can be captured only within BSE). Accordingly, we have also worked on the TDDFT kernel, f_{xc}, implementation in transition space. The new algorithm projects f_{xc} in transition space by performing integrals in reciprocal space. Compared to the previous implementation, where integrals were performed in real-space, this gives a speed-up of roughly one order of magnitude. An example of magnons dispersion taken for a recently submitted manuscript is provided [19].

Yambopy calculators for new features. `Yambopy` entered its 0.4.x version (leading to v1 release once documentation and test-suite will be completed). Beside a relevant streamlining of the code, several new features were added, including: (*i*) A new class to perform the unfolding of a band structure from a supercell (e.g., including a defect) to the pristine unit cell; (*ii*) A complete rewriting of the database loading electronic wavefunctions, which now supports spinors, fast Fourier transform to real space (e.g., for 3D visualization via `.xsf` or `.cube` output files), and the application of crystal symmetry operations; (*iii*) Extended support for post-processing of spin-polarized calculations for magnetic systems; (*iv*) A new suite of tools for the analysis of nonlinear optical spectra and advanced management of databases produced by nonlinear optics calcula-

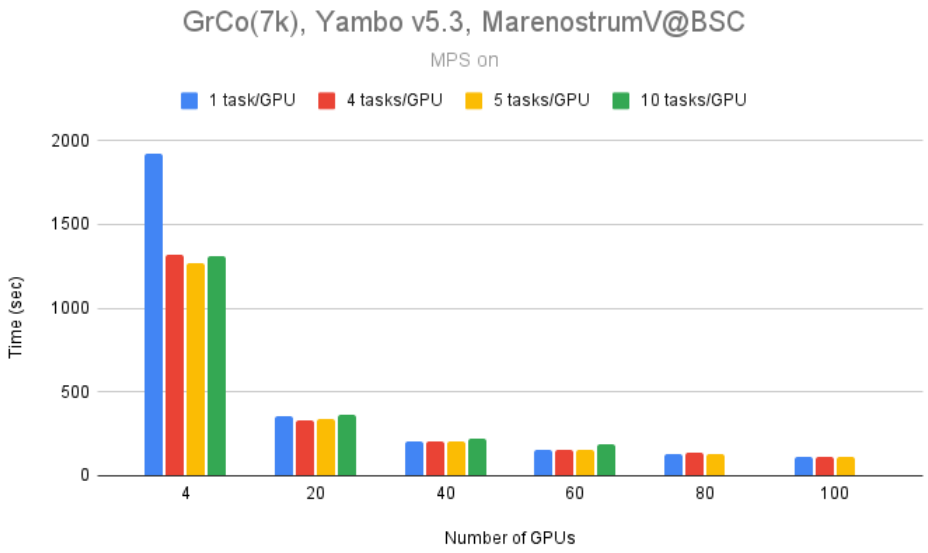

Figure 16: GrCo-small benchmark, performance tests using MPS with Yambo (v5.3.0) on MareNostrum 5 at BSC. The same strong scaling test was run changing the ratio of MPI tasks per GPU.

tions; (v) An interface with the new auxiliary electron-phonon code, `LetzElPhC` (see also deliverable D2.4 for a detailed description) in order to load, convert and analyse electron-phonon coupling matrix elements.

Electron and exciton coupling to phonons. The new workflow for the efficient calculation of exciton-phonon coupling properties is now fully working (see also deliverable D2.4). It relies on an interface between DFPT (QUANTUM ESPRESSO) and BSE (YAMBO) methodologies and databases, ensuring gauge consistency of the wave functions and the possibility to make full use of the crystal symmetries in the spaces of both electron and phonon momenta. This allows for the calculation of exciton-phonon coupling matrix elements and related spectroscopic properties (indirect optical spectra, Raman spectra, temperature-dependent scattering lifetimes) as a simple post-processing. The computational cost of an exciton-phonon calculation is then just the sum of the symmetry-enabled DFPT and BSE underlying calculations.

GW calculations on top of a DFT+U ground state. This capability relies on the reconstruction of the *bare* (i.e., non-interacting) electronic Hamiltonian inside YAMBO in order to exclude double counting of the "U" correction. It can also be applied to calculations performed with hybrid functionals. This feature currently works for three-dimensional, non-spinorial systems (including magnetic systems), and work is ongoing to include also two-dimensional and spinorial support.

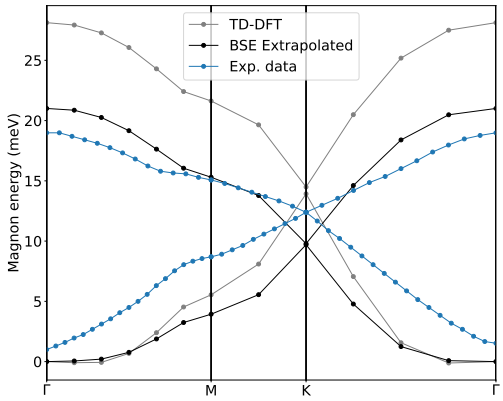

Figure 17: From preprint [19]. Comparison between magnon dispersions in CrI_3 computed with the TD-DFT and BSE methods. The experimental magnon dispersion of CrI_3 from Ref. [20] is included in blue.

7 Conclusions

This second release report emphasizes the transition of MAX flagship codes from consolidated, production-grade HPC tools to advanced, exascale-enabled applications capable of addressing frontier scientific challenges.

All codes have undergone performance optimization both intranode, with improved or additional porting to homogeneous and heterogeneous HPC machines, and internode, in such a way to enable more effective exploitation of large scale HPC partitions. Software engineering best practices, adopted since the early phases of MAX , have been further consolidated and extended. Among these, the investment in reproducible deployment pipelines across diverse EuroHPC systems through Spack, EasyBuild, and EESSI, as well as the measurement and collection of benchmarking data, stressing the deployment and performance levels achieved by the MAX codes.

A common thread across all contributions is the strategic balance between incremental refinement of existing features and the exploration of novel capabilities tailored for exascale platforms. BigDFT has demonstrated a robust, asynchronous QM/MM workflow operating across CPU-GPU partitions, with application focused around insight into drug resistance mechanisms via first-principles analyses of large biomolecular complexes. FLEUR significantly extended its scientific scope through phonon simulations in 2D materials and a modular linear algebra interface, while showcasing impressive scalability and preparedness for the JUPITER supercomputer. QUANTUM ESPRESSO progressed in code modularization and GPU support via OpenACC, establishing a foundation for unified development streams and better integration of performance-oriented branches. SIESTA introduced advanced real-time TD-DFT features and enhanced QM/MM capabilities. YAMBO delivered substantial performance increase in the core response function kernel, while also obtaining gains through a modular solver interface, GPU-enabled diagonalization paths, and preconditioned iterative solvers, with ongoing support for novel physics such as phonon-exciton coupling.

Beyond the individual technical achievements, this release testifies to the collaborative and modular vision of MAX : fostering a shared ecosystem where various develop-

ments in each code inspire and benefit the others. Looking ahead, the next development cycle will be shaped by three main strategic axes:

- Workflow integration and automation: Continued investment in interoperable workflows and meta-schedulers to facilitate multi-code applications across exascale platforms.
- Precision-performance trade-offs: Extension of mixed-precision methodologies, adaptive solvers, and algorithmic reductions to meet power and memory constraints at scale.
- Scientific showcases: Deeper engagement with domain-specific grand challenges — ranging from quantum materials to biological interfaces — using real-world applications to guide and validate code development.

References

[1] Handy, N. C. & Schaefer III, H. F. On the evaluation of analytic energy derivatives for correlated wave functions. *J. Chem. Phys.* **81**, 5031–5033 (1984).

[2] Hutter, J. Excited state nuclear forces from the tamm–dancoff approximation to time-dependent density functional theory within the plane wave basis set framework. *J. Chem. Phys.* **118**, 3928 (2003).

[3] Jin, Y., Wen-zhe Yu, V., Govoni, M., Xu, A. C. & Galli, G. Excited state properties of point defects in semiconductors and insulators investigated with time-dependent density functional theory. *J. Chem. Theory Comput.* **19**, 8689–8705 (2023).

[4] Cerezo, J. & Santoro, F. Fcclasses3, vibrationally-resolved spectra simulated at the edge of the harmonic approximation. *J. Comput. Chem.* (2022).

[5] Santoro, F. & Cerezo, J. Fcclasses3 , a code for vibronic calculations, version 3.0.4, 2025. URL http://www.iccom.cnr.it/en/fcclasses.

[6] Artacho, E. & O'Regan, D. D. Quantum mechanics in an evolving hilbert space. *Phys. Rev. B* **95**, 115155 (2017).

[7] Wittemeier, N., Papior, N., Brandbyge, M., Zanolli, Z. & Ordejón, P. Quantum transport with spin orbit coupling: New developments in transiesta (2025). URL https://arxiv.org/abs/2501.16162. 2501.16162.

[8] Henes, M., Lockbaum, G. J. & et al. Acs chem. biol. *ACS Chem. Biol.* **14** (2019).

[9] Ratcliff, L. E., Dawson, W. & et al. J. chem. phys. *J. Chem. Phys.* **152** (2020).

[10] Jung, J., Mori, T. & et al. Wires comput. mol. sci. *WIREs Comput. Mol. Sci.* **5** (2015).

[11] Dawson, W., Mohr, S. & et al. J. chem. theory comput. *J. Chem. Theory Comput.* **16** (2020).

[12] Genovese, L., Dawson, W. & et al. J. chem. phys. *J. Chem. Phys.* **158** (2023).

[13] Dawson, W., Beal, L., Ratcliff, L. E. & et al. Electron. struct. *Electron. Struct.* **6** (2024).

[14] Ozaki, K., Ogita, T., Oishi, S. & Rump, S. M. Error-free transformations of matrix multiplication by using fast routines of matrix multiplication and its applications. *Numerical Algorithms* **59**, 95–118 (2011).

[15] Ydiago library (2024). URL https://github.com/yambo-code/Ydiago.

[16] Milev, P. *et al.* Performances in solving the bethe-salpeter equation with the yambo code (2025). URL https://arxiv.org/abs/2504.10096. 2504.10096.

[17] Alvarruiz, F., Mellado-Pinto, B. & Roman, J. E. Variants of thick-restart lanczos for the bethe-salpeter eigenvalue problem (2025). URL https://arxiv.org/abs/2503.20920. 2503.20920.

[18] Spallanzani, N. Yambo on arm architectures. Tech. Rep. 0470568, Istituto Nanoscienze, CNR, Italy (2024). DOI: 10.5281/zenodo.14260222.

[19] Esquembre-Kučukalić, A. *et al.* Magnons in chromium trihalides from *ab initio* bethe-salpeter equation (2025). URL https://arxiv.org/abs/2502.06598. 2502.06598.

[20] Chen, L. *et al.* Topological spin excitations in honeycomb ferromagnet cri_3. *Phys. Rev. X* **8**, 041028 (2018).